

Automatic Customization of OS and Runtime Stacks*

Martin Schulz (LLNL, schulzm@llnl.gov, main contact)
David Lowenthal (U. of Arizona, dkl@cs.arizona.edu)

Exascale systems will be complex not only in their hardware architecture, but also in their software ecosystem. Further, it is expected that for an application to successfully reach exascale, the overall system has to be designed to best exploit the specific characteristics for that application and that conventional OS and runtime efforts will not be sufficient in many cases. This has spawned a wide range of co-design as well as programming and/or execution model research. However, fusing these varying requirements into a single software ecosystem that can support this variety is difficult and can easily lead to one of two naive solutions: the creation of large shared runtimes that span all areas or individual stovepipes for each application area and/or programming approach. Neither solution is desirable: the latter case is not sustainable beyond a very small number of applications, while the former is likely to lead to "super" runtime and OS solutions that can do everything, but nothing well and will fail to utilize resources efficiently. In order to overcome this dilemma, we need a new, third approach.

1 Research Approach

We need techniques that help us to automatically specialize a single, but highly modular OS/R stack to the characteristics of individual applications. We can achieve this by extracting application characteristics and then using them to adjust the OS/R stack either statically once for an application area, statically during the application compilation, or even dynamically at runtime. In all cases, we can create a solution tailored to the application and its needs, but not more. This opens up several optimization opportunities, including:

Memory Footprint Reduction: by only including the required components we can reduce the size of the OS/R stack and its associated resources. For example, applications that do not need parts of the MPI standard can run on an MPI subset, applications that do not require threading can be built without thread support in the OS, or task based programming models can replace the scheduler with their own.

Optimization through Specialization: after removing individual elements in the OS/R stack, the remaining parts can be streamlined. For example, if OS level threading is not required, libraries don't need to be thread-safe; if communication patterns are known, any message or communication engine can be simplified by eliminating the need for general matching; or if shared data structures in PGAS systems can be detected and isolated at compile-time, the memory management in the OS can be simplified.

Breaking Walls Between Models: most future systems will require multiple forms of parallelism, typically arranged in some kind of hierarchy. In many cases this requires the co-existence of multiple execution models or subsets of them, which offers the possibility to merge their runtimes and execution approaches. For example, in programs that use both Pthreads for outer parallelism and OpenMP for inner parallelism (such as within libraries), it may be possible to unify the threading runtime; or in codes combining Chapel and UPC through different submodules, the required RMA support can be fused and made fully interoperable to the point that each model can transparently access data managed by the other model.

Optimization Through Parameter Tuning: a flexible and modular stack will provide a wide range of performance tuning knobs (buffer sizes, scheduling parameters, memory management policies, etc.). Their optimal setting often depends on the application and is a promising target for auto-tuning approaches both at runtime and compile time.

*This work was partially performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344 (LLNL-ABS-564233).

To implement this approach, three major challenges need to be addressed: extracting the information out of the application, either at compile or at runtime; identifying the modules in the OS/R that can be adjusted, removed, or fused; and transforming the set of given OS/R modules into a minimalistic OS and runtime system. In a first step, the necessary application requirements and characteristics can either be extracted or detected automatically through coarse grain static analysis. Longer term, this can be replaced with a more fine grain analysis of both the application, its dependent libraries, and the elements of the underlying communication scheme. Similarly, early experiments can use manual analysis to drive simple subsetting of OS/R modules, while later work can target automatic transformation and cross-module integrations.

The result is an adaptive OS and runtime approach that specializes itself to the requirements of a particular application without creating stovepipe solutions. This will not only enable the reuse of OS/R modules across application areas, but also help guide subsetting of OS/R modules, exposing additional performance optimization opportunities.

2 Existing Work

The idea of lean OS solutions is not new and has lead to the idea of micro-kernels, which later have been adapted to HPC in the form of lightweight compute kernels, such as Catamount, the BlueGene Compute Node Kernel CNK, or Cray’s Compute Node Linux CNL. However, none of these existing infrastructure automatically generates, or even tries to generate, customized operating systems for current systems. Systems like K42 [1], SmartApps [3], Daises [4], or the approach by Krintz and Wolski [2, 5] provide some reconfigurability, but either do not provide a solution on how to adjust or are limited to selected OS parameters. Most importantly all these approaches do not include customization in the runtime layer.

For runtime layers, multiple efforts are on the way to allow better interoperability between different programming or execution models. Examples are work in the MPI forum to provide more support for hybrid models; or efforts (e.g., in ROSE or LLVM) to concurrently support multiple DSLs on a single system. However, none of these projects target the ability to reconfigure or to extract only relevant subsets.

3 Assessment

Challenges: This work poses three major OS/R challenges: it targets the potential explosion of required runtime support for the wide range of execution and programming models expected at exascale; it enables performance improvements by supporting a reduced, right-sized OS/R stack; and it requires a new, highly modular thinking in any OS/R design.

Maturity: The idea builds on successful research, incl. in the DOE funded FASTOS efforts, on lightweight kernels, which have shown the advantages of reduced OS/R environments. Additionally, hybrid models combining multiple execution and/or programming models are an active area of research, but often fall short in terms of compose-ability and interference reduction.

Uniqueness: While the general principles could apply to any compute environment, its necessity and urgency will be first felt when pushing the limits in scaling applications and their varying ecosystems to large scale. It is therefore not likely that other areas or communities will produce the required advances for the problems targeted here.

Novelty: As described above, the work leverages successful projects in the area of micro- and lightweight kernels, but expands them in many areas. It provides a combined view of OS and runtime, and adds the concept of automatic, application specific specialization.

Applicability: If successful, the results of this project will likely have a significant trickle down effect and will be useful for smaller scale solutions, like Petascale in a rack systems as well as enterprise sized clusters.

Effort: Investigating this approach is likely to be a multi-year effort and should include research teams from a wide range of communities, including programming models and DSLs (to drive the customization); runtime, OS, and communication layers (to implement the customization); and performance analysis and modeling (to guide the customization process).

References

- [1] O. Krieger, M. Auslander, B. Rosenberg, R. W. Wisniewski, J. Xenidis, D. Da Silva, M. Ostrowski, J. Appavoo, M. Butrico, M. Mergen, A. Waterland, and V. Uhlig. K42: building a complete operating system. *SIGOPS Oper. Syst. Rev.*, 40(4):133–145, 2006.
- [2] C. Krintz and R. Wolski. Using phase behavior in scientific application to guide linux operating system customization. In *NSF Next Generation Software Workshop*, 2005.
- [3] L. Rauchwerger and N. M. Amato. Smartapps: middle-ware for adaptive applications on reconfigurable platforms. *SIGOPS Oper. Syst. Rev.*, 40(2):73–82, 2006.
- [4] P. J. Teller and S. R. Seelam. Insights into providing dynamic adaptation of operating system policies. *SIGOPS Oper. Syst. Rev.*, 40(2):83–89, 2006.
- [5] L. Youseff, R. Wolski, and C. Krintz. Linux kernel specialization for scientific application performance. TR CS200529, UC Santa Barbara, Nov. 2005.